

CLAIMS

- 1 1. An apparatus comprising:
 - 2 at least one processor;
 - 3 a memory coupled to the at least one processor;
 - 4 a plurality of object oriented classes residing in the memory, at least one of the
 - 5 plurality of object oriented classes including state data that indicates a protected class;
 - 6 a state/domain checker residing in the memory and executed by the at least one
 - 7 processor, the state/domain checker performing a plurality of checks when each of the
 - 8 plurality of object oriented classes is loaded, the plurality of checks determining whether
 - 9 a class being loaded accesses at least one protected class, and if so, determining whether
 - 10 the class being loaded is authorized to access the at least one protected class, and
 - 11 generating an exception if the class being loaded is not authorized to access the at least
 - 12 one protected class.
- 1 2. The apparatus of claim 1 wherein the state/domain checker further performs at least
 - 2 one runtime check when a method that may reference a dynamically defined class is
 - 3 invoked and when a function is invoked that could potentially access a method on one or
 - 4 more of the plurality of classes.

1 3. The apparatus of claim 2 wherein the at least one runtime check includes a check to
2 determine whether a Java reflection method is invoked by a referencing class on a
3 referenced class, and if a Java reflection method is invoked by the referencing class, and
4 if the referenced class implements a private domain interface, and if the referencing class
5 does not implement a system state interface, generating an error.

1 4. The apparatus of claim 2 wherein the at least one runtime check includes a check to
2 determine whether a Java Native Interface (JNI) function is invoked by an external
3 program to access a protected class, and if the external program invokes a JNI function to
4 access a protected class, and the external program is not running in system state,
5 generating an error.

1 5. The apparatus of claim 1 wherein a class is a protected class if the class is defined as a
2 private domain class or a system state class.

1 6. The apparatus of claim 1 wherein the plurality of checks includes a check during class
2 verification that determines whether a class being verified implements a private domain
3 interface or a system state interface, and if the class being verified implements a private
4 domain interface or a system state interface, and if the class being verified is not included
5 in a catalog of allowed classes, generating an error.

1 7. The apparatus of claim 6 wherein the catalog of allowable classes is generated during
2 a build process that packages the plurality of classes together into an installable form.

1 8. The apparatus of claim 1 wherein the plurality of checks includes a check during class
2 preparation that determines whether a class being prepared has a superclass, and if the
3 class being prepared has a superclass, and if the superclass implements a private domain
4 interface or a system state interface, and if the class being prepared does not implement at
5 least the same private domain interface or system state interface as the superclass,
6 generating an error.

1 9. The apparatus of claim 1 wherein the plurality of checks includes a check during class
2 resolution that determines whether a class being resolved to by a referencing class
3 implements a private domain interface, and if the class being resolved to by the
4 referencing class implements the private domain interface, and if the referencing class
5 does not implement a system state interface, generating an error.

1 10. The apparatus of claim 9 wherein the check during class resolution is performed
2 before runtime when a class is loaded.

1 11. The apparatus of claim 9 wherein the check during class resolution is performed at
2 runtime when a method on the class being resolved to is invoked.

1 12. An apparatus comprising:
2 at least one processor;
3 a memory coupled to the at least one processor;
4 a plurality of Java classes residing in the memory, at least one of the plurality of
5 Java classes including state data that indicates a protected class;
6 a Java Virtual Machine (JVM) residing in the memory and executed by the at least
7 one processor;
8 a state/domain checker residing in the memory and executed by the at least one
9 processor, the state/domain checker performing the following checks:
10 a first check during class verification that determines whether a class being
11 verified implements a private domain interface or a system state interface, and if
12 the class being verified implements a private domain interface or a system state
13 interface, and if the class being verified is not included in a catalog of allowed
14 classes that is generated during a JVM build process that packages the plurality of
15 classes together into an installable form, throwing an exception;
16 a second check during class preparation that determines whether a class
17 being prepared has a superclass, and if the class being prepared has a superclass,
18 and if the superclass implements a private domain interface or a system state
19 interface, and if the class being prepared does not implement at least the same
20 private domain interface or system state interface as the superclass, throwing an
21 exception;
22 a third check during class resolution that determines whether a class being
23 resolved to by a referencing class implements a private domain interface, and if
24 the class being resolved to by the referencing class implements the private domain
25 interface, and if the referencing class does not implement a system state interface,
26 throwing an exception;

(claim 12 continued)

27 a fourth check to determine whether a Java reflection method is invoked
28 by a referencing class on a referenced class, and if a Java reflection method is
29 invoked by the referencing class, and if the referenced class implements a private
30 domain interface, and if the referencing class does not implement a system state
31 interface, throwing an exception; and

32 a fifth check to determine whether a Java Native Interface (JNI) function is
33 invoked by a program external to the JVM to access a protected class at runtime,
34 and if the program invokes a JNI function to access a protected class, and the
35 program is not running in system state, throwing an exception.

1 13. A method for creating and enforcing protected system level Java code comprising the
2 steps of:

3 loading a plurality of Java classes, each of the plurality of Java classes that is
4 protected including state data that indicates a protected class;
5 performing a plurality of checks when each of the plurality of Java classes is
6 loaded, the plurality of checks determining whether the class being loaded accesses at
7 least one protected class, and if so, determining whether the class being loaded is
8 authorized to access the at least one protected class, and generating an exception if the
9 class being loaded is not authorized to access the at least one protected class.

1 14. The method of claim 13 further comprising the step of performing at least one
2 runtime check when a method that may reference a dynamically defined class is invoked
3 and when a function is invoked that could potentially access a method on one or more of
4 the plurality of classes.

1 15. The method of claim 14 wherein the at least one runtime check includes a check to
2 determine whether a Java reflection method is invoked by a referencing class on a
3 referenced class, and if a Java reflection method is invoked by the referencing class, and
4 if the referenced class implements a private domain interface, and if the referencing class
5 does not implement a system state interface, generating an error.

1 16. The method of claim 14 wherein the at least one runtime check includes a check to
2 determine whether a Java Native Interface (JNI) function is invoked by an external
3 program to access a protected class, and if the external program invokes a JNI function to
4 access a protected class, and the program is not running in system state, generating an
5 error.

1 17. The method of claim 13 wherein a class is a protected class if the class is defined as a
2 private domain class or a system state class.

1 18. The method of claim 13 wherein the plurality of checks includes a check during class
2 verification that determines whether a class being verified implements a private domain
3 interface or a system state interface, and if the class being verified implements a private
4 domain interface or a system state interface, and if the class being verified is not included
5 in a catalog of allowed classes, generating an error.

1 19. The method of claim 18 wherein the catalog of allowable classes is generated during
2 a JVM build process that packages the plurality of classes together into an installable
3 form.

1 20. The method of claim 13 wherein the plurality of checks includes a check during class
2 preparation that determines whether a class being prepared has a superclass, and if the
3 class being prepared has a superclass, and if the superclass implements a private domain
4 interface or a system state interface, and if the class being prepared does not implement at
5 least the same private domain interface or system state interface as the superclass,
6 generating an error.

1 21. The method of claim 13 wherein the plurality of checks includes a check during class
2 resolution that determines whether a class being resolved to by a referencing class
3 implements a private domain interface, and if the class being resolved to by the
4 referencing class implements the private domain interface, and if the referencing class
5 does not implement a system state interface, generating an error.

1 22. The method of claim 21 wherein the check during class resolution is performed
2 before runtime when a class is loaded by a Java Virtual Machine (JVM).

1 23. The apparatus of claim 21 wherein the check during class resolution is performed at
2 runtime when a method on the class being resolved to is invoked.

1 24. A method for creating and enforcing protected system level Java code comprising the
2 steps of:

3 running a Java Virtual Machine (JVM);

4 the JVM loading a plurality of Java classes, each of the plurality of Java classes
5 that is protected including state data that indicates a protected class;

6 performing a first check during class verification that determines whether a class
7 being verified implements a private domain interface or a system state interface, and if
8 the class being verified implements a private domain interface or a system state interface,
9 and if the class being verified is not included in a catalog of allowed classes that is
10 generated during a JVM build process that packages the plurality of classes together into
11 an installable form, throwing an exception;

12 performing a second check during class preparation that determines whether a
13 class being prepared has a superclass, and if the class being prepared has a superclass, and
14 if the superclass implements a private domain interface or a system state interface, and if
15 the class being prepared does not implement at least the same private domain interface or
16 system state interface as the superclass, throwing an exception;

17 performing a third check during class resolution that determines whether a class
18 being resolved to by a referencing class implements a private domain interface, and if the
19 class being resolved to by the referencing class implements the private domain interface,
20 and if the referencing class does not implement a system state interface, throwing an
21 exception;

22 performing a fourth check to determine whether a Java reflection method is
23 invoked by a referencing class on a referenced class at runtime, and if a Java reflection
24 method is invoked by the referencing class, and if the referenced class implements a
25 private domain interface, and if the referencing class does not implement a system state
26 interface, throwing an exception; and

(claim 24 continued)

27 performing a fifth check to determine whether a Java Native Interface (JNI)
28 function is invoked by a program external to the JVM to access a protected class at
29 runtime, and if the program invokes a JNI function to access a protected class, and the
30 program is not running in system state, throwing an exception.

IBM Docket No. ROC9-2000-0095-US1 34

1 25. A method for building a computer program that includes system level code, the
2 method comprising the steps of:
3 generating Java source code for a plurality of object oriented classes, each of the
4 plurality of object oriented classes that is protected including state data defined in the
5 Java source code that indicates a protected class;
6 identifying from the Java source code for each object oriented class which of the
7 plurality of object oriented classes are protected;
8 compiling the Java source code for the plurality of object oriented classes, thereby
9 creating a plurality of class files that correspond to the plurality of object oriented classes;
10 creating a catalog of allowable classes, the catalog including all protected classes;
11 compiling source code for a Java Virtual Machine (JVM) to produce an
12 executable JVM;
13 creating installable media that includes the executable JVM, the catalog of
14 allowable classes, and the class files.

1 26. A program product comprising:
2 a state/domain checker that performs a plurality of checks when each of a plurality
3 of object oriented classes is loaded, the plurality of checks determining whether a class
4 being loaded accesses at least one protected class, and if so, determining whether the
5 class being loaded is authorized to access the at least one protected class, and generating
6 an exception if the class being loaded is not authorized to access the at least one protected
7 class; and
8 signal bearing media bearing the state/domain checker.

1 27. The program product of claim 26 wherein said signal bearing media comprises
2 recordable media.

1 28. The program product of claim 26 wherein said signal bearing media comprises
2 transmission media.

1 29. The program product of claim 26 wherein the state/domain checker further performs
2 at least one runtime check when a method that may reference a dynamically defined class
3 is invoked and when a function is invoked that could potentially access a method on one
4 or more of the plurality of classes.

1 30. The program product of claim 29 wherein the at least one runtime check includes a
2 check to determine whether a Java reflection method is invoked by a referencing class on
3 a referenced class, and if a Java reflection method is invoked by the referencing class, and
4 if the referenced class implements a private domain interface, and if the referencing class
5 does not implement a system state interface, generating an error.

1 31. The program product of claim 29 wherein the at least one runtime check includes a
2 check to determine whether a Java Native Interface (JNI) function is invoked by an
3 external program to access a protected class, and if the external program invokes a JNI
4 function to access a protected class, and the external program is not running in system
5 state, generating an error.

1 32. The program product of claim 26 wherein a class is a protected class if the class is
2 defined as a private domain class or a system state class.

1 33. The program product of claim 26 wherein the plurality of checks includes a check
2 during class verification that determines whether a class being verified implements a
3 private domain interface or a system state interface, and if the class being verified
4 implements a private domain interface or a system state interface, and if the class being
5 verified is not included in a catalog of allowed classes, generating an error.

1 34. The program product of claim 33 wherein the catalog of allowable classes is
2 generated during a build process that packages the plurality of classes together into an
3 installable form.

1 35. The program product of claim 26 wherein the plurality of checks includes a check
2 during class preparation that determines whether a class being prepared has a superclass,
3 and if the class being prepared has a superclass, and if the superclass implements a
4 private domain interface or a system state interface, and if the class being prepared does
5 not implement at least the same private domain interface or system state interface as the
6 superclass, generating an error.

1 36. The program product of claim 26 wherein the plurality of checks includes a check
2 during class resolution that determines whether a class being resolved to by a referencing
3 class implements a private domain interface, and if the class being resolved to by the
4 referencing class implements the private domain interface, and if the referencing class
5 does not implement a system state interface, generating an error.

1 37. The program product of claim 36 wherein the check during class resolution is
2 performed before runtime when a class is loaded.

1 38. The program product of claim 36 wherein the check during class resolution is
2 performed at runtime when a method on the class being resolved to is invoked.

1 39. A program product comprising:
2 (A) a plurality of Java classes, at least one of the plurality of Java classes
3 including state data that indicates a protected class;
4 (B) a Java Virtual Machine (JVM) executable program;
5 (C) a state/domain checker that performs the following checks:
6 (C1) a first check during class verification that determines whether a class
7 being verified implements a private domain interface or a system state interface,
8 and if the class being verified implements a private domain interface or a system
9 state interface, and if the class being verified is not included in a catalog of
10 allowed classes that is generated during a JVM build process that packages the
11 plurality of classes together into an installable form, throwing an exception;
12 (C2) a second check during class preparation that determines whether a
13 class being prepared has a superclass, and if the class being prepared has a
14 superclass, and if the superclass implements a private domain interface or a
15 system state interface, and if the class being prepared does not implement at least
16 the same private domain interface or system state interface as the superclass,
17 throwing an exception;
18 (C3) a third check during class resolution that determines whether a class
19 being resolved to by a referencing class implements a private domain interface,
20 and if the class being resolved to by the referencing class implements the private
21 domain interface, and if the referencing class does not implement a system state
22 interface, throwing an exception;
23 (C4) a fourth check to determine whether a Java reflection method is
24 invoked by a referencing class on a referenced class at runtime, and if a Java
25 reflection method is invoked by the referencing class, and if the referenced class
26 implements a private domain interface, and if the referencing class does not
27 implement a system state interface, throwing an exception;

(claim 39 continued)

28 (C5) a fifth check to determine whether a Java Native Interface (JNI)
29 function is invoked by a program external to the JVM to access a protected class
30 at runtime, and if the program invokes a JNI function to access a protected class,
31 and the program is not running in system state, throwing an exception; and
32 (D) signal bearing media bearing the plurality of Java classes, the JVM executable
33 program, and the state/domain checker.

1 40. The program product of claim 39 wherein said signal bearing media comprises
2 recordable media.

1 41. The program product of claim 39 wherein said signal bearing media comprises
2 transmission media.
